Engineer

Dépôt

Stockage conservant les versions d'un ensemble de fichiers

Ce que c'est

Un dépôt, repository en anglais et presque toujours abrégé en repo dans la conversation, est l'endroit où un système de gestion de versions conserve tout l'historique d'un projet : chaque état enregistré de chaque fichier, l'auteur et la raison de chaque changement, et les noms donnant accès à cet historique. Les fichiers de travail que vous modifiez en sont une vue ; le dépôt est tout ce qui se trouve derrière.

Git définit précisément le terme dans son glossaire : « Un ensemble de références, accompagné d'une base d'objets contenant tous les objets accessibles depuis ces références. » Les deux parties méritent une explication. La base d'objets contient le contenu. Chaque version d'un fichier, chaque liste du contenu d'un répertoire et chaque commit est stocké sous forme d'objet, nommé par une empreinte de son contenu. Le livre Git décrit cela comme un système de fichiers adressé par le contenu : un stockage clé-valeur où la clé est calculée à partir de la valeur. Deux fichiers identiques ne forment qu'un objet, et toute modification, aussi petite soit-elle, produit un nouveau nom. Les références sont la partie humaine : branches et tags, noms courts pointant chacun vers un commit et donnant un point d'entrée aux lecteurs.

La plupart des dépôts dans lesquels on travaille ont une arborescence de travail attachée. Les fichiers ouverts dans l'éditeur se trouvent dans le répertoire du projet et le dépôt lui-même vit à côté, dans un répertoire caché `.git`. Un dépôt nu, bare, n'a aucune arborescence de travail. Son nom de répertoire se termine conventionnellement par `.git`. Il ne contient que l'historique et constitue la forme conservée sur un serveur, car personne n'y modifie de fichiers : on y pousse des modifications et on en récupère. Le glossaire Git le précise : les éditeurs de dépôts publics proposent généralement des dépôts nus. Le livre Git montre comment en créer un depuis un projet existant avec `git clone --bare my_project my_project.git`.

Le clonage révèle la conception de Git. Un clone reçoit presque toutes les données du serveur : par défaut, chaque version de chaque fichier sur l'ensemble de l'historique du projet l'accompagne. Chaque clone est donc un dépôt complet capable de remplacer le serveur si celui-ci est perdu. Les exceptions sont limitées mais importantes, et le livre Git les nomme : les hooks côté serveur et les réglages similaires restent sur place.

Deux variantes reviennent souvent. Un dépôt peu profond, créé par clonage avec `--depth`, ne garde que les commits récents. Le glossaire dit que les parents plus anciens sont « cautérisés », et l'historique peut être approfondi plus tard par une récupération. Un sous-module est un dépôt contenu dans un autre, avec son propre historique, que le dépôt extérieur fixe à un commit précis.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un dépôt ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des changements testés, publiés et réversibles, pour livrer aux clients un mardi après-midi plutôt qu'un vendredi soir tard.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Déploiement SaaS GIS automatisé : GitHub Actions — déploiement, données et reporting automatisés

Déploiement SaaS GIS automatisé : GitHub Actions Contactez-nous →

Pourquoi c'est important

Un dépôt est la mémoire d'un projet. Lorsqu'une ligne de code semble incorrecte, l'historique dit quand elle est arrivée, dans quel commit et avec quelle explication. Cette mémoire ne vaut que ce que valent les commits écrits. C'est pourquoi les équipes soucieuses de leur historique imposent des règles aux messages et les font respecter par des hooks plutôt que par la bonne volonté.

C'est aussi l'unité sur laquelle travaillent la plupart des outils. Les accès s'accordent par dépôt, les hooks s'exécutent par dépôt et l'intégration continue commence généralement depuis un dépôt. Définir ses limites revient à délimiter ce que couvrent une revue, un contrôle et une permission.

Les dépôts gardent ce qu'on leur donne. Un commit supprimant un fichier ajoute un nouvel état : il laisse tous les états précédents à leur place, dans chaque clone qui les détient déjà. Cette permanence est la raison d'être de la gestion de versions et fonctionne dans les deux sens : une ancienne version reste récupérable, mais aussi un mot de passe commité par erreur.

Enfin, un dépôt peut contenir bien plus que du code applicatif. Définitions d'infrastructure, documentation et configuration y gagnent les mêmes choses : un historique, une étape de revue et un moyen de reconstruire un état connu. Un serveur dont la configuration vit dans un dépôt peut être reconstruit par quelqu'un qui ne l'a jamais vu.

À quoi cela ressemble en pratique

Quatre dépôts construits en dix-huit mois par une personne partagent une convention de commits : préfixe conventionnel indiquant le type de changement, périmètre, sujet d'une longueur inférieure à la limite fixée, aucun emoji ni ligne d'attribution en fin de message. Un hook de message de commit, présent dans chaque dépôt où du travail est réalisé, rejette toute violation. La convention devient ainsi une propriété du dépôt plutôt que de la personne qui commite. L'historique montre ensuite la nature du travail. Au stade d'implémentation de juillet 2026 décrit dans la documentation, l'historique du plus grand dépôt comptait 156 commits de fonctionnalités, 133 corrections, 128 de documentation, 81 d'entretien, 26 refactorisations, 20 de style, 5 de performance et 3 de tests. Le hook s'arrête avant la qualité : il vérifie le format, pas le contenu.

Pour l'infrastructure propre à l'entreprise, les services déployés accessibles au public et la configuration d'autres composants, l'objectif était de les consigner dans un dépôt plutôt que de les garder en tête. L'implémentation consignée comprenait 12 065 lignes de YAML, réparties en 19 playbooks et 34 rôles. Ces nombres décrivent ce stade, pas le dépôt actuel. L'automatisation des sauvegardes et du courrier y est configurée, avec une activation opérationnelle distincte. L'infrastructure peut être reproduite depuis le dépôt : un hôte appartient à un groupe correspondant à un niveau d'infrastructure, dont les variables déclarent ses rôles, et un hôte convergé signale zéro changement. Chaque modification prend plus de temps qu'une édition directe sur le serveur, et un play de vérification préalable a ensuite été ajouté avant la convergence.

Un dépôt garde aussi ce qu'il a reçu pendant que l'attention était ailleurs. Du travail pour une autre entreprise a survécu à une restructuration du dépôt, notamment un journal de travail contenant des identifiants de connexion actifs en clair, resté dans l'arborescence sans être détecté par aucun contrôle pendant l'essentiel de l'historique du dépôt. Rien de tout cela ne s'exécutait : c'est précisément pourquoi cela a survécu. La réponse a été une règle de périmètre en tête des instructions du dépôt, l'infrastructure d'une seule entreprise et rien

d'autre, ainsi qu'une quarantaine plutôt qu'une suppression. L'historique reste lisible et le détecteur de secrets, le contrôle du coffre-fort et le contrôle des emojis continuent de lire les fichiers en quarantaine.

Où les problèmes surviennent

- **Un secret commité une fois survit à sa suppression.** Retirer le fichier au commit suivant le laisse dans le précédent et dans tous les clones faits depuis. Traitez ces informations de connexion comme publiques : renouvelez-les d'abord, puis décidez si réécrire l'historique justifie les perturbations pour tous les détenteurs d'une copie.
- **Un push rencontre une branche extraite.** Pousser vers un dépôt non nu dont la branche courante est celle mise à jour désynchroniserait son arborescence de travail et son historique. Git refuse donc ce push par défaut au moyen de `receive.denyCurrentBranch`. Un dépôt recevant des push devrait être nu.
- **Un clone peu profond répond sur une partie de l'historique.** Un clone créé avec `--depth` n'emporte que les commits récents. `git log`, `git blame` et tout script déduisant une version du dernier tag ne voient donc que cette tranche. Les tâches de build et de publication nécessitant le passé devraient récupérer tout l'historique.
- **Un hook local s'arrête au clone suivant.** Les hooks côté client vivent dans `.git/hooks`, et le livre Git précise qu'un clone ne les copie pas. Une politique imposée uniquement à cet endroit cesse le jour d'un nouveau clonage. Versionnez les scripts de hook dans le dépôt et faites pointer `core.hooksPath` vers eux, ou imposez la règle côté serveur avec `pre-receive`.
- **Les gros binaires gonflent chaque clone indéfiniment.** Puisqu'un clone emporte chaque version de chaque fichier, chaque ancienne copie d'un gros binaire remplacé reste dans l'historique et tous les nouveaux clones les téléchargent toutes.
- **Les fichiers que rien n'exécute ne sont pas lus.** Une tâche désactivée, un inventaire périmé ou un journal de travail reste dans chaque clone, mais aucun test, build ou déploiement n'y touche : il échappe donc à la revue. Le contenu du dépôt devrait correspondre aux responsabilités du projet. Tout le reste devrait être retiré ou isolé là où les contrôles peuvent encore le voir.

Pour aller plus loin

- gitglossary : dépôt — la définition de Git, à côté du dépôt nu, de l'arborescence de travail, du dépôt peu profond et du sous-module.
- Obtenir un dépôt Git — les deux façons d'en obtenir un, en partant d'un répertoire ou en clonant, et ce qu'un clone récupère.

- Objets Git — stockage du contenu dans la base d'objets sous l'empreinte de ce contenu.
- gitrepository-layout — rôle de chaque fichier et répertoire à l'intérieur d'un dépôt.
- Installer Git sur un serveur — transformer un dépôt existant en dépôt nu et le partager.
- Hooks Git — hooks côté client et serveur, et raison pour laquelle un clone ne récupère pas ceux du client.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un dépôt ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/repository/